Chapter 3

Mano's Basic Computer

REVIEW

- **We introduced registers**
- **We discussed the kinds of Transfer, Arithmetic, Logic and Shift Microoperations**
- **We discussed general aspects of CPU, Memory and Bus architectures**

- **In this chapter we introduce a more complicated <u>basic computer and show how its operation can be specified with register transfer statements</u>**

CONSIDERING THE NEXT PROBLEM IN DESIGN

A computer organization involves combining everything we have learned to date into a single integrated unit

- What is a computer?

Von Neuman refers to a computer as a "stored program digital computer"

- What is a program?
  - What is an instruction?
  - How are instructions executed?

GOALS

We conclude our lecture series by considering computer organization

- Combining the CPU, Memory and Bus architectures

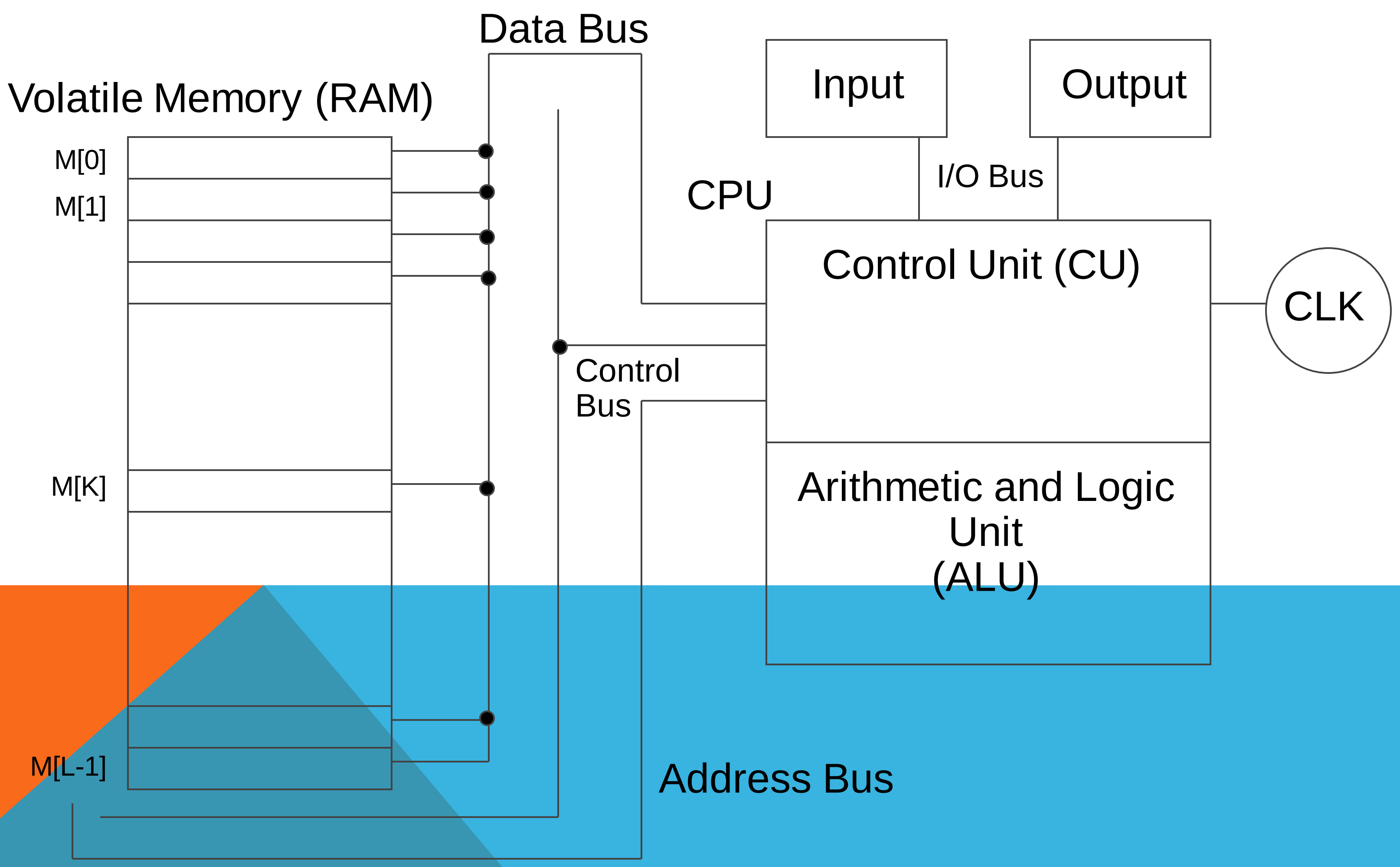
We introduce the concept of an instruction

- Instruction design and architecture
- Microoperation sequencing (timing, control)
- Roles of different registers

We will follow a model of a virtual/logical computer organization adapted from M. Mano (Computer System Architecture, 3d Edition)

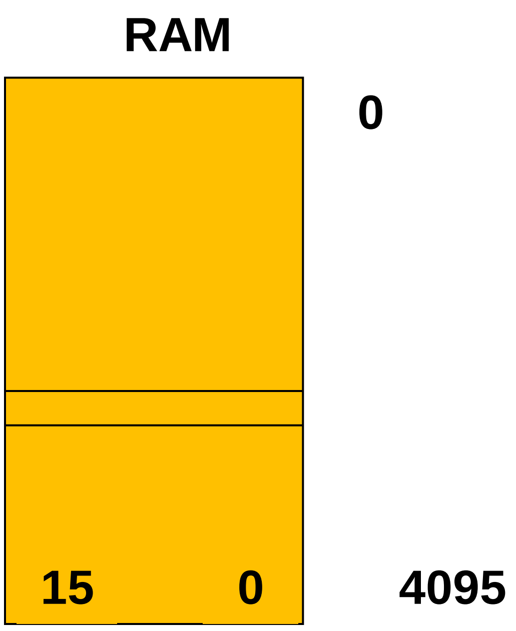
COMPUTER – HIGH LEVEL VIEW

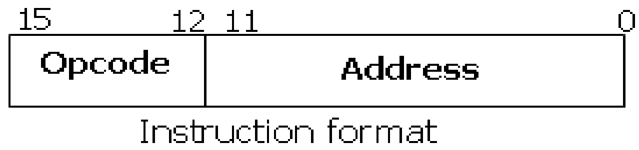
CPU, Memory and Bus architectures with interface to I/O.

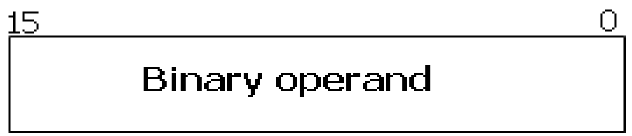


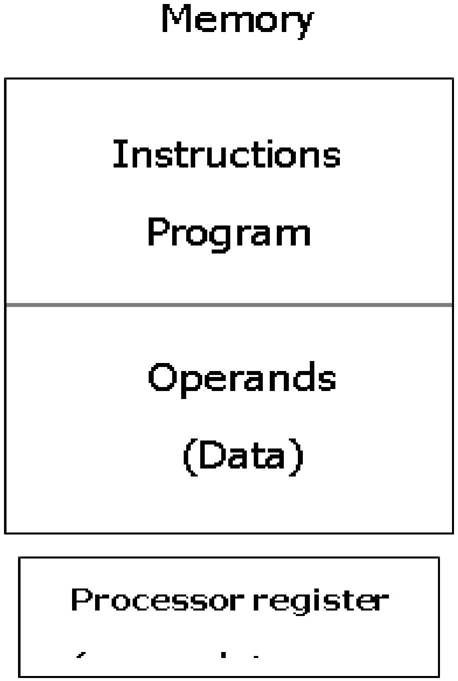
MANO'S COMPUTER SPECIFICATION

- Mano has a Memory (RAM) of 4096 words.
- This requires an Address bus of 12 bits.
- Each word is 16 bits long.
- The Data bus carries exactly 1 word of data between Memory and CPU.
- It has 8 registers.
- Input and Output is defined using the ASCII code of length 8 bits.



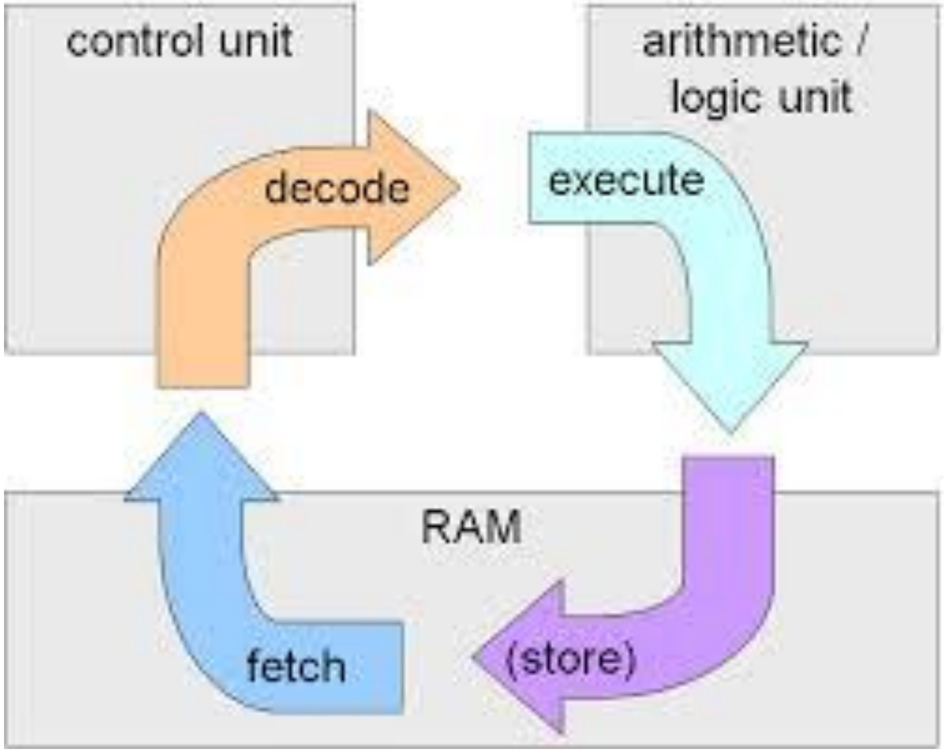






- Instruction is stored in one 16-bit memory word,
- It consists 4 bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand (data used).
- The control reads a 16-bit instruction from the program portion of memory. It then executes the operation specified by the operation code.
- Computers that have a single-processor register usually assign to it the name Accumulator and label it AC.

FETCH – DECODE – EXECUTE CYCLE



THE MANO MODEL OF THE CPU REGISTERS

CPU registers used in the model (Mano), categorized by <u>length</u>:

- PC :: Program counter (hold address of next inst. – 12 bits)
- AR :: Address register (holds address for memory– 12 bits)

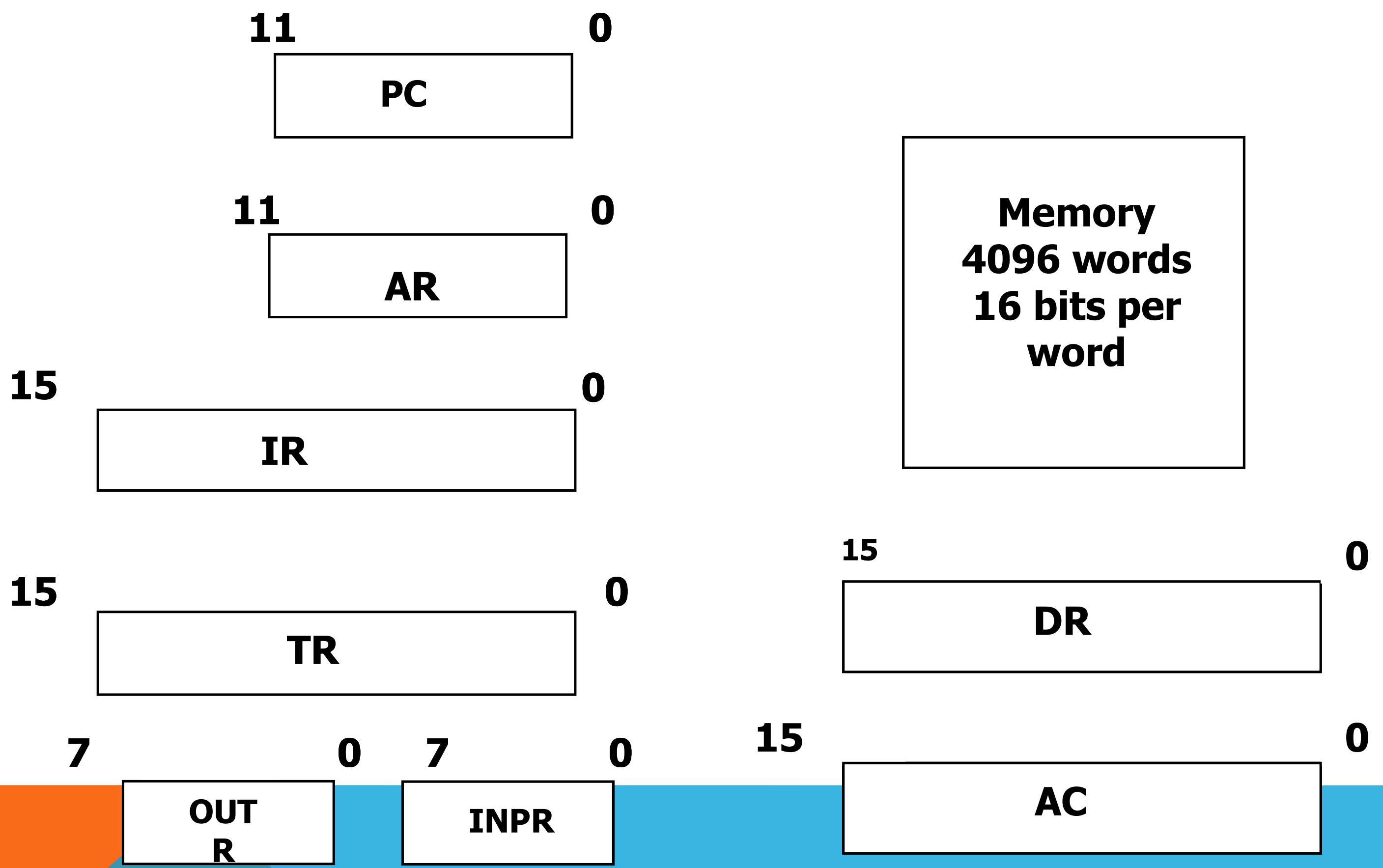
- DR :: Data register (holds memory operand– 16 bits)
- AC :: Accumulator (processor register holds data – 16 bits)
- IR :: Instruction register (holds instruction code – 16 bits)
- TR :: Temporary register (holds temporary data – 16 bits)

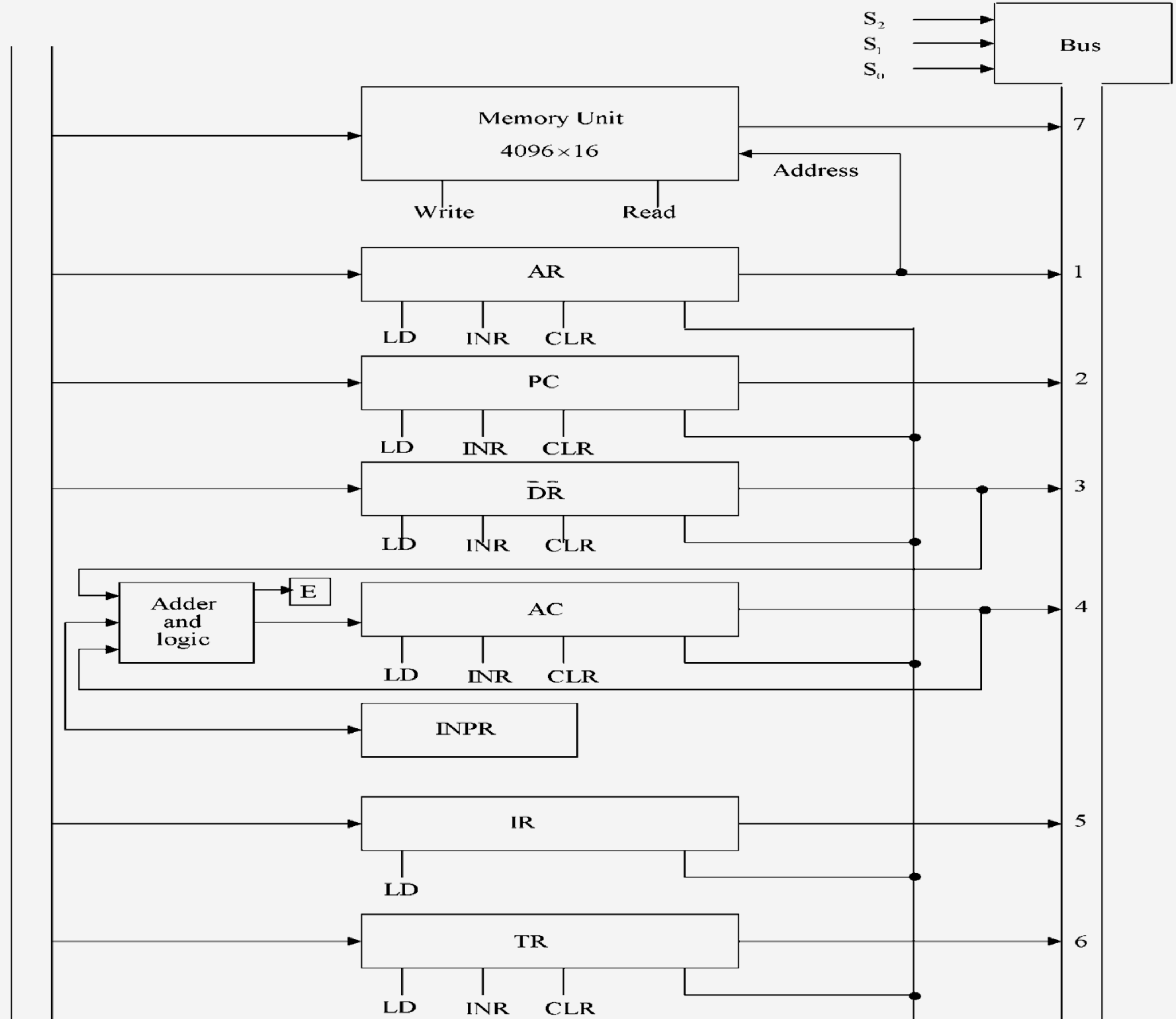
- INR :: Input buffer register (holds input ASCII data – 8 bits)
- OUTR :: Output buffer register (holds output ASCII data – 8 bits)

SCR :: Sequence counter register (4 bits)

E, R :: Single bit flip-flops (flag/utility, interrupt)

BASIC COMPUTER REGISTERS AND MEMORY




S2
S1
S0
Bus
Memory Unit
4096×16
Address
Write
Read
7
AR
LD
INR
CLR
1
PC
LD
INR
CLR
2
DR
LD
INR
CLR
3
Adder
and
logic
E
AC
LD
INR
CLR
4
INPR
IR
LD
5
TR
LD
INR
CLR
6

STORED PROGRAMS

- ❖ A stored program is a set of instructions and data expressed in binary language, stored in non-volatile (ie. disk storage) memory

Programs can be executed only from Memory.

- Thus, a program must be loaded from disk to RAM in order to execute it.
- Loaded programs are called *processes*.
- Individual *instructions* must be transferred to the CPU where they are executed, using data that must be obtained from either CPU registers or RAM

- ❖ A process is executed by executing each individual instruction that, collectively, fulfill the intentions of the programmer.

COMPUTER INSTRUCTION

- **A computer instruction is a binary code that specifies a sequence of microoperations for the computer.**
- **Instruction codes together with data are stored in memory (RAM).**
- **The computer reads each instruction from memory and places it in a control register.**
- **The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.**
- **Operations such as: add, subtract, multiply, shift, and complement.**
- The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.

RELATIONSHIP BETWEEN A COMPUTER OPERATION AND A MICROOPERATION

- **An operation is <u>part of an instruction stored in computer memory</u>. It is a binary code that tells the computer to perform a specific operation.**
- **For every operation code, the control issues a sequence of microoperations needed for the hardware implementation of the specified operation.**
- **An instruction code must specify not only the operation but also the registers (by binary code) or the memory words (by address) where the operands are to be found, as well as the register or memory word where the result is to be stored.**

INSTRUCTION ARCHITECTURE

The list of all instructions engineered for a computer's CPU is called the *instruction set*.

To distinguish each instruction, they are assigned a <u>unique</u> numeric code

- Can be done in many ways
- Does not need to be ordinal (ie. starting from 0 and running contiguously)
- Can be partially ordinal and partially hierarchical
  - Mano's approach

Instructions must be capable of referencing Memory and/or CPU addresses in order to cause data to be transferred and operated on.

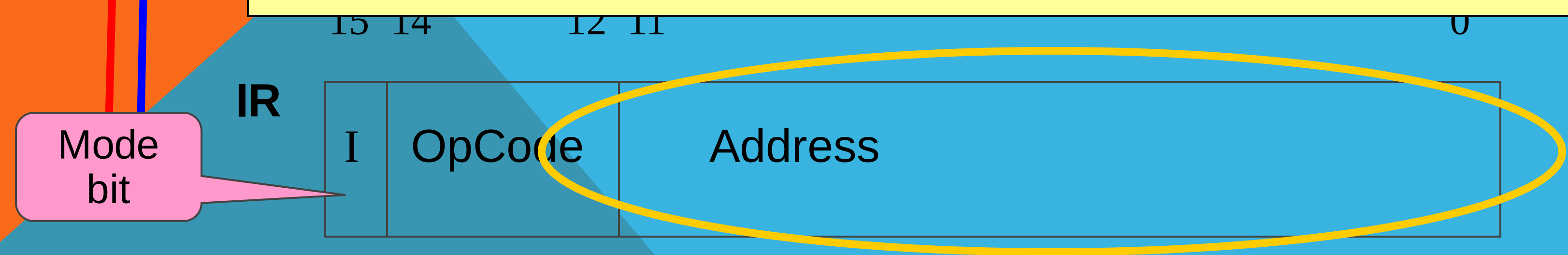
INSTRUCTION FORMAT

Instructions usually consist of:

- Operation code (opcode) part
- Address part
- Addressing Mode part
  - Direct addressing – data found at the specified address
  - Indirect addressing – data found at the address found at the specified address (pointer concept)

The number of memory referenced operands varies from computer to computer.

Mano simplifies the instruction architecture to permit zero or one memory address field. In the case of Indirect addressing mode there are two memory accesses, but only one reference.



MANO’S INSTRUCTION FORMAT

- **In Mano’s Computer, since the memory contains 4096 (= 2^{12}) words, we needs 12 bit to specify which memory address this instruction will use**
- **Bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing)**
- **3 bits for the instruction’s opcode (possible 8 operations)**

MANO'S COMPUTER INSTRUCTION FORMAT

	14 12	11 0
I	opcode	Address

(a)Memory –reference

15	14 12	11 0
0	1 1 1	Register operation

(Opcode=111,I=0)

(b)Register –reference instruction

15	14 12	11 0
1	1 1 1	I/O operation

(Opcode=111,I=1)

(c)Input-Output instruction

INSTRUCTION HIERARCHY

Mano's instruction set consists of 25 instructions:

- Instruction
  - OpCode (000-110)
    - Direct (I=0)
    - Indirect (I=1)
  - OpCode (111)
    - Direct (I=0)
    - Indirect (I=1)

OpCode (000-110), Direct (I=0)

4 bit code	
0	AND
1	ADD
2	LDA
3	STA
4	BUN
5	BSA
6	ISZ

OpCode (000-110), Indirect (I=1)

4 bit code	
8	AND
9	ADD
A	LDA
B	STA
C	BUN
D	BSA
E	ISZ

7

OpCode (111), Direct (I=0)

16 bit code	
7800	CLA
7400	CLE
7200	CMA
7100	CME
7080	CIR
7040	CIL
7020	INC
7010	SPA
7008	SNA
7004	SZA
7002	SZE
7001	HLT

12

OpCode (111), Indirect (I=1)

16 bit code	
F800	INP
F400	OUT
F200	SKI
F100	SKO
F080	ION
F040	IOF

6

Symbol	*Hex Code* *I = 0*	*I = 1*	*Description*
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

INSTRUCTION SET COMPLETENESS

Selection of instructions should span a variety of applications suitable to support programming

- Arithmetic, logical and shift instructions
- Instructions for moving data to and from memory and CPU registers
- Program control instructions, instructions that check status conditions
- Input and Output instructions

INSTRUCTION SET COMPLETENESS

Set of instructions using which user can construct machine language programs to evaluate any computable function.

- Instruction Types

Functional Instructions
- Arithmetic, logic, and shift instructions
- ADD, CMA, INC, CIR, CIL, AND, CLA (other than ADD/AND?)

Transfer Instructions
- Data transfers between the main memory and the processor registers
- LDA, STA

Control Instructions
- Program sequencing and control
- BUN, BSA, ISZ

Input/Output Instructions
- Input and output
- INP, OUT

INSTRUCTION CYCLE

- **In Computer, a machine instruction is executed in the following cycle:**
  1. Fetch an instruction from memory
  2. Decode the instruction and calculate effective address (EA)
  3. Read the EA from memory if the instruction has an indirect address (Fetch operand)
  4. Execute the instruction

- **Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction.**
- **This process continues indefinitely unless a <u>HALT</u> instruction is encountered.**

FETCH AND DECODE CYCLE

Microoperations for the fetch and decode phases can be specified by the following register transfer statements:

T_0: AR←PC

Fetch from memory the current instruction (which address is in PC)

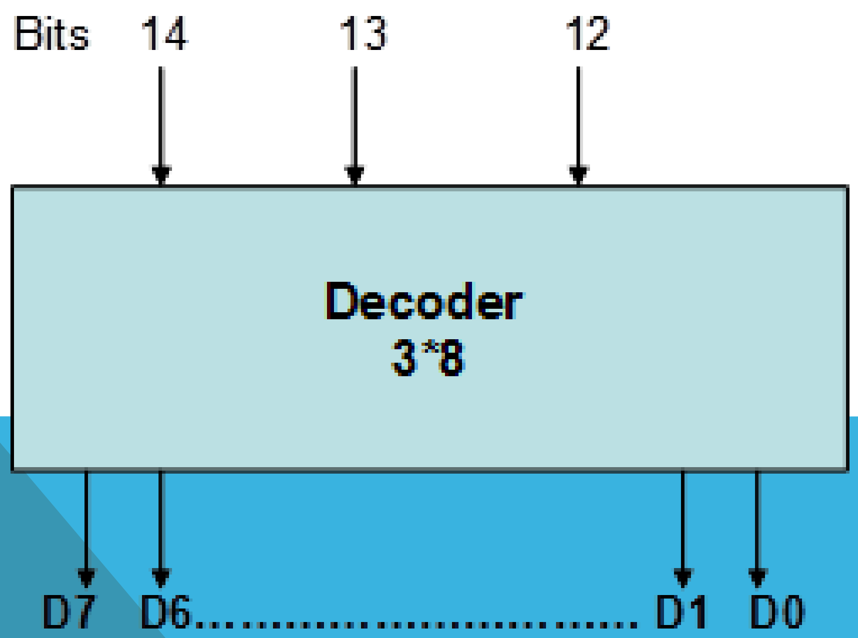
T_1: IR←M[AR] ; PC←PC+1

T_3 : D_0 ,...,D_7 Decode IR(12-14) ,AR←IR(0-11) ,I←IR(15)

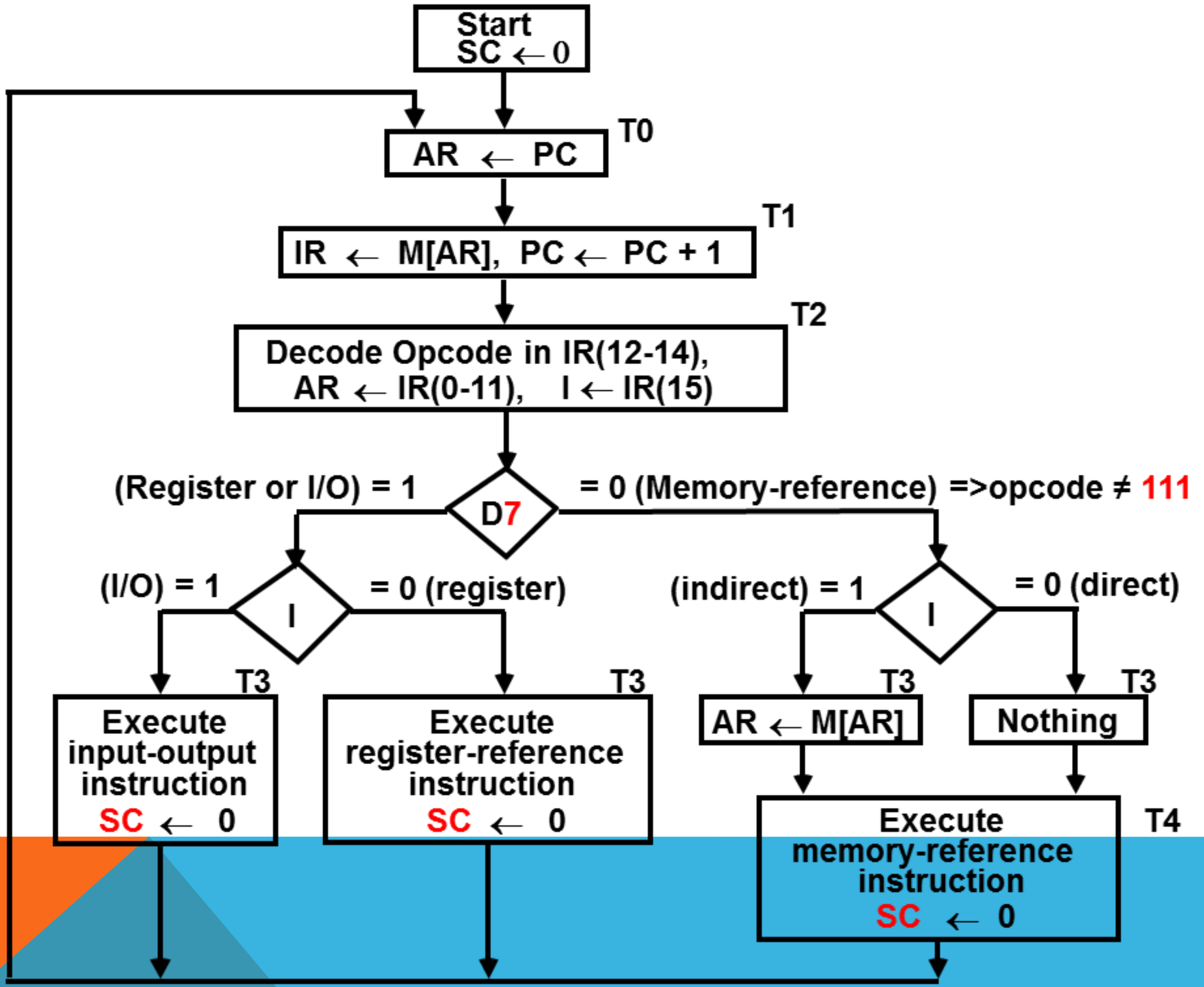
Decode instruction found now in IR

DETERMINE THE TYPE OF INSTRUCTION

- We use bits 12 , 13, 14 as input to the decoder to determine the type of instruction
- Only one of the decoder outputs = 1 at certain combination (i.e. operation) of the inputs.

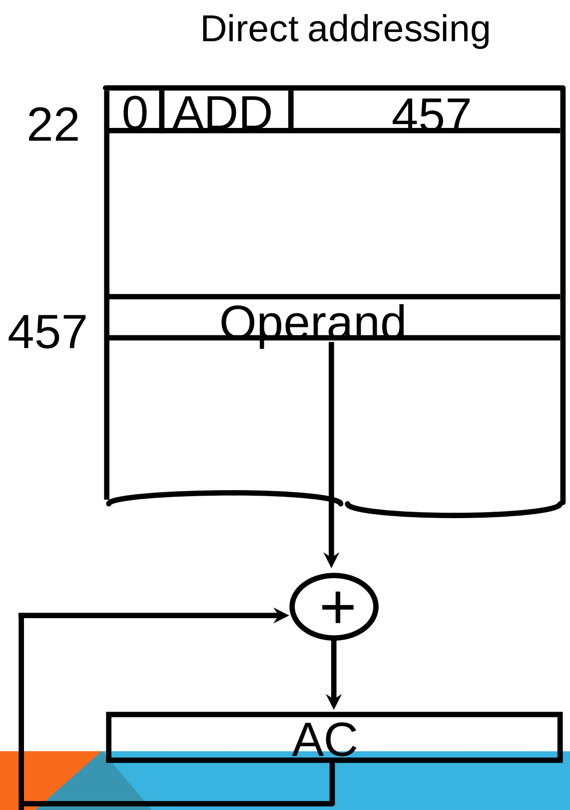


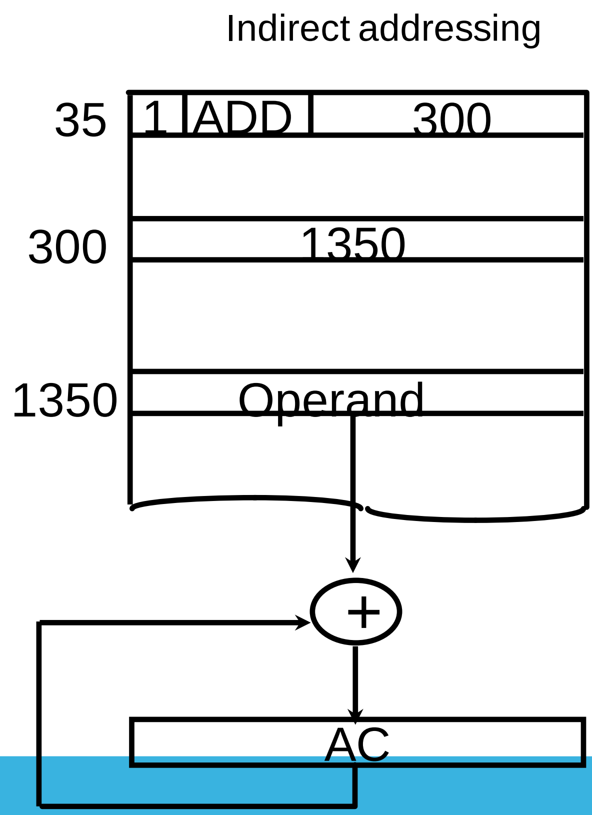
DETERMINE THE TYPE OF INSTRUCTION



D'_7IT_3:	AR ← M[AR]
$D'_7I'T_3$:	Nothing
$D_7I'T_3$:	Execute a register-reference instr.
D_7IT_3:	Execute an input-output instr.

DIFFERENCE BETWEEN DIRECT AND INDIRECT ADDRESSING





1. MEMORY-REFERENCE INSTRUCTIONS:

Symbol	Operation Decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow C_{OUT}$
LDA (load to accumulator)	D_2	$AC \leftarrow M[AR]$
STA (Store from accumulator)	D_3	$M[AR] \leftarrow AC$
BUN (branch unconsitionally)	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR+1$
ISZ	D_6	$M[AR] \leftarrow M[AR]+1$, IF $M[AR]+1=0$ then

D0: AND: AND TO AC

- **This is an instruction that performs the AND logic operation on pairs of bits in *AC* and the memory word specified by the effective address found in (AR) now.**
- **We can not work on the memory location directly, we have to bring the data to DR (Data Register) first.**
- **The result of the operation is transferred to *AC*.**
- **The microoperations that execute this instruction are:**

D_0T_4: DR← M[AR]

D_0T_5: AC← AC ^DR, SC← 0

D1: ADD: ADD TO AC

- **This instruction adds the content of the memory word specified by the effective address found in (AR) now to the value of *AC*.**
- **We can not work on the memory location directly, we have to bring the data to DR (Data Register) first.**
- **The result of the operation is transferred to *AC*.**
- **The microoperations that execute this instruction are:**

D_1T_4: $DR \leftarrow M[AR]$

D_1T_5: $AC \leftarrow AC + DR,\ E \leftarrow Cout,\ SC \leftarrow 0$

D2: LDA: LOAD TO AC

- **This instruction transfers the memory word specified by the effective address found in (AR) now to AC.**
- **We can not work on the memory location directly, we have to bring the data to DR (Data Register) first.**
- **The microoperations that execute this instruction are:**

$$D_2T_4: \quad DR \leftarrow M[AR]$$

$$D_2T_5: \quad AC \leftarrow DR, \; SC \leftarrow 0$$

D3: STA: STORE AC

- **This instruction stores the content of AC into the memory word specified by the effective address found in (AR) now.**

- **The microoperation that execute this instruction is:**

$$D_3T_4: \quad M[AR] \leftarrow AC \text{ , } SC \leftarrow 0$$

D4: BUN: BRANCH UNCONDITIONALLY

- **This instruction transfers the program to the instruction specified by the effective address found in (AR) now.**
- **The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.**
- **The effective address from AR is transferred through the common bus to PC.**
- **The microoperation that execute this instruction is:**

$$D_4T_4: \quad PC \leftarrow AR \quad , SC \leftarrow 0$$

D5: BSA: BRANCH AND SAVE RETURN ADDRESS

- **This instruction is useful for branching to a portion of the program called a subroutine or procedure.**
- **When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address found in (AR) now.**
- **The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine.**
- **The microoperation that execute this instruction is:**

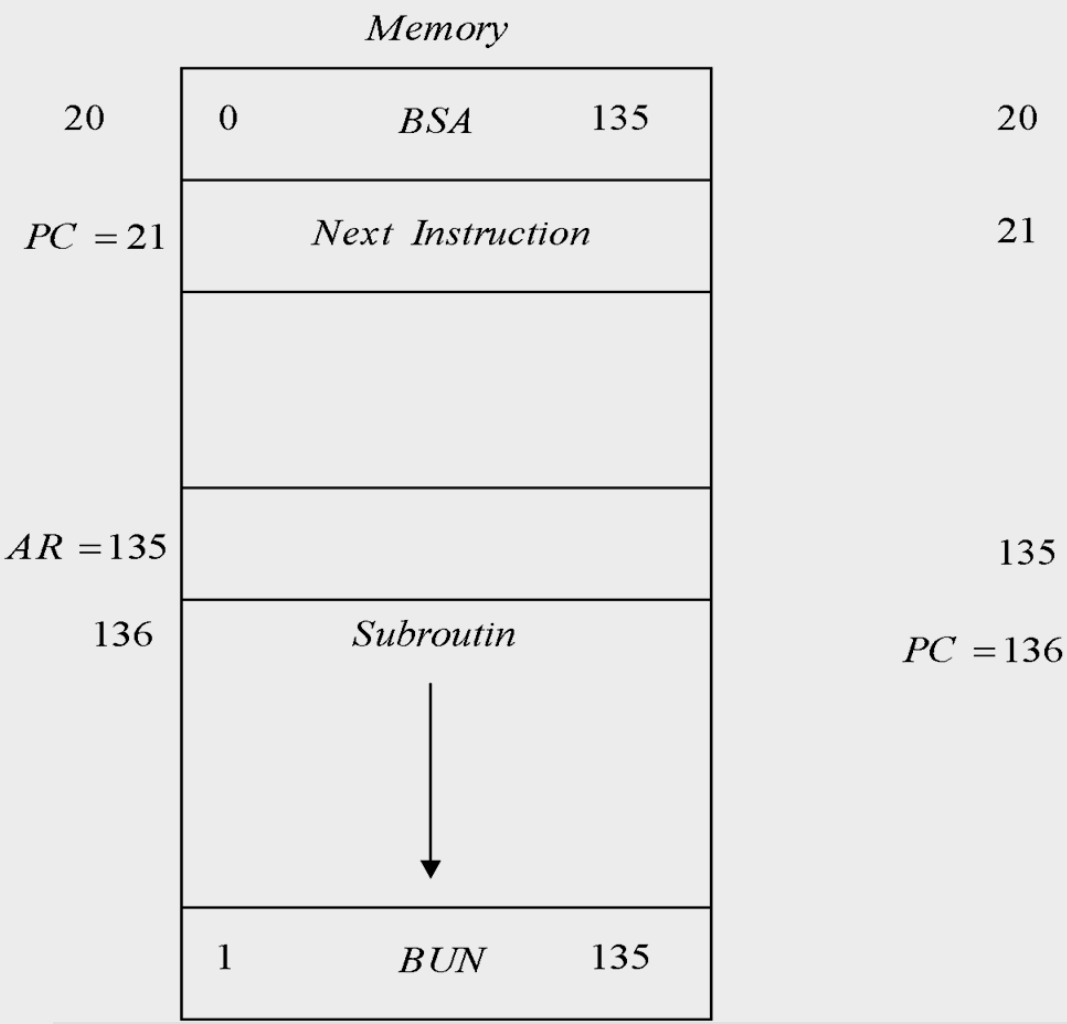
D_4T_4: $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$,

D_5T_4: $PC \leftarrow AR$, $SC \leftarrow 0$

EXAMPLE ON BSA

- **The BSA instruction is assumed to be in memory at address 20**
- **The *I bit* is 0 and the address part of the instruction has the binary equivalent of 135.**
- **After the fetch and decode phases, PC contains 21, which is the address of the next instruction in the program (referred to as the return address).**
- **AR holds the effective address 135.**
- **The BSA instruction performs the following numerical operation:**

$$M[135] \leftarrow 21 \text{ , } PC \leftarrow 135 + 1 = 136$$



A) Memory, Pc and AR at time T4

B) memory, PC and AR after execution

Figure 3.8 Example of BSA instruction execution.

D6: ISZ: INCREMENT AND SKIP IF ZERO

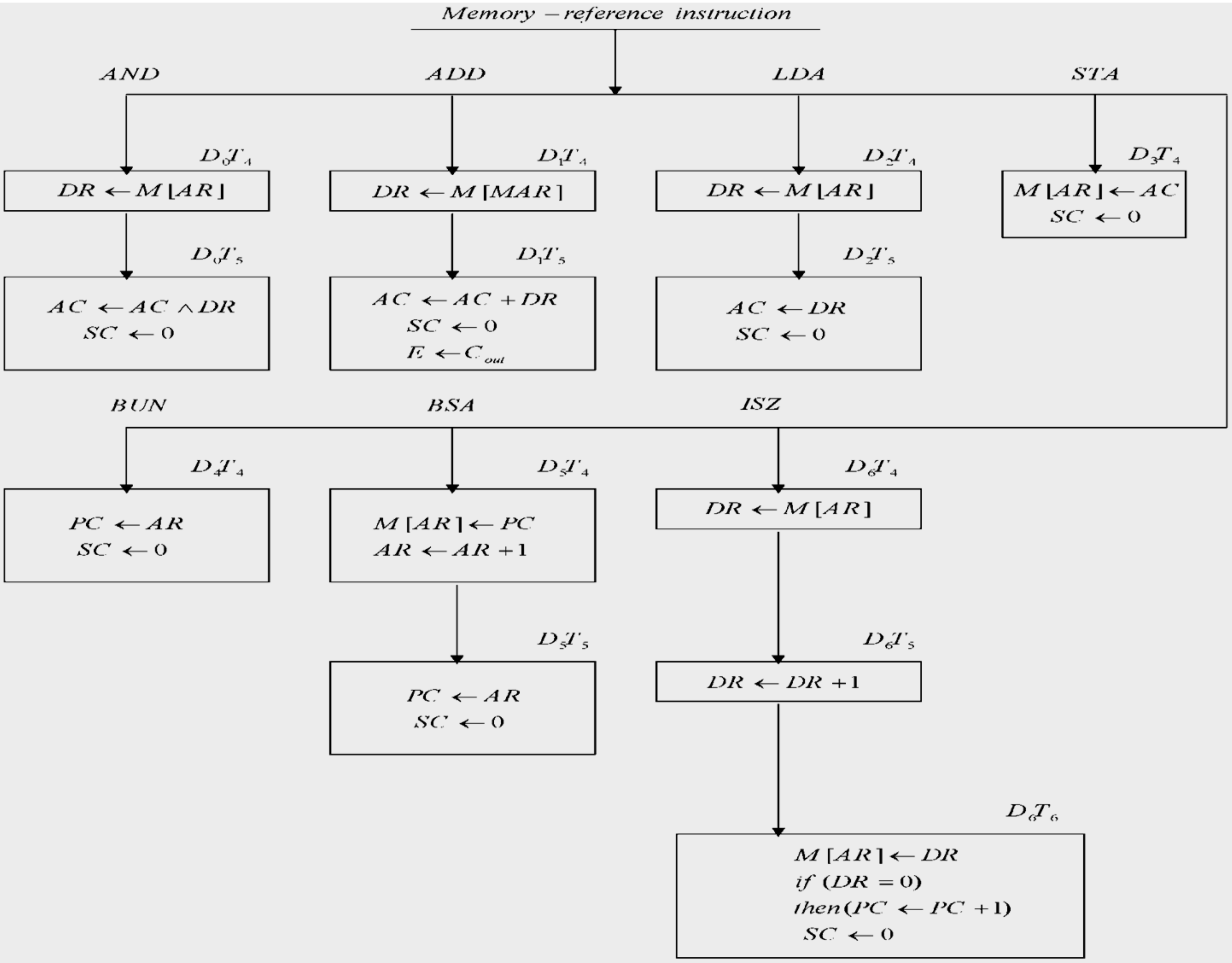
- **This instruction increments the word specified by the effective address found in (AR) now, and if the incremented value is equal to 0, PC is incremented by 1.**
- **The programmer usually stores a negative number (in 2's complement) in the memory word (AR).**
- **As this negative number is repeatedly incremented by one, it eventually reaches the value of zero.**
- **At that time PC is incremented by one in order to skip the next instruction in the program.**
- **Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.**

- The microoperations that execute this instruction are:

D_6T_4: $DR \leftarrow M[AR]$

D_6T_5: $DR \leftarrow DR+1$

D_6T_6: $M[AR] \leftarrow DR$, if$(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$



Memory – reference instruction
AND
ADD
LDA
STA
D_0T_4
$DR \leftarrow M[AR]$
D_0T_5
$AC \leftarrow AC \wedge DR$
$SC \leftarrow 0$
D_1T_4
$DR \leftarrow M[MAR]$
D_1T_5
$AC \leftarrow AC + DR$
$SC \leftarrow 0$
$E \leftarrow C_{out}$
D_2T_4
$DR \leftarrow M[AR]$
D_2T_5
$AC \leftarrow DR$
$SC \leftarrow 0$
D_3T_4
$M[AR] \leftarrow AC$
$SC \leftarrow 0$
BUN
BSA
ISZ
D_4T_4
$PC \leftarrow AR$
$SC \leftarrow 0$
D_5T_4
$M[AR] \leftarrow PC$
$AR \leftarrow AR + 1$
D_5T_5
$PC \leftarrow AR$
$SC \leftarrow 0$
D_6T_4
$DR \leftarrow M[AR]$
D_6T_5
$DR \leftarrow DR + 1$
D_6T_6
$M[AR] \leftarrow DR$
if $(DR = 0)$
then $(PC \leftarrow PC + 1)$
$SC \leftarrow 0$

REGISTER REFERENCE INSTRUCTIONS

12 INSTRUCTIONS, STARTS WITH 0111

Hex	Mnemonic	Binary coding	Meaning
7800	CLA	0111100000000000	CLear Accumulator
7400	CLE	0111010000000000	CLear E
7200	CMA	0111001000000000	CoMplement Accumulator
7100	CME	0111000100000000	CoMplement E
7080	CIR	0111000010000000	CIrcular shift Right
7040	CIL	0111000001000000	CIrcular shift Left
7020	INC	0111000000100000	INCrement AC
7010	SPA	0111000000010000	Skip if Positive Accumulator
7008	SNA	0111000000001000	Skip if Negative Accumulator
7004	SZA	0111000000000100	Skip if Zero Accumulator
7002	SZE	0111000000000010	Skip if Zero E
7001	HLT	0111000000000001	HaLT the computer

REGISTER REFERENCE INSTRUCTIONS

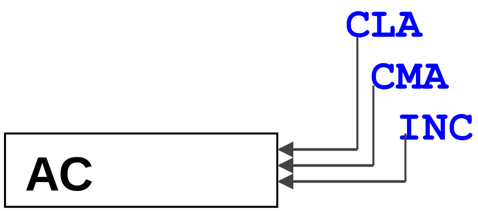
- Within this instruction subset, we group the instructions by which register is affected

Hex	Mnemonic	
7800	CLA	AC affected 8 instructions
7200	CMA	
7080	CIR	
7040	CIL	
7020	INC	
7010	SPA	
7008	SNA	
7004	SZA	
7400	CLE	E affected 3 instructions
7100	CME	
7002	SZE	
7001	HLT	Control affected

REGISTER REFERENCE INSTRUCTIONS

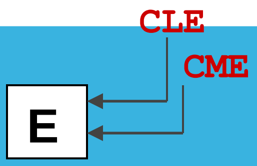
Clear, Complement or Increment : AC register.

Hex	Mnemonic	RTL
7800	CLA	AC = 0
7200	CMA	AC = ~ AC
7020	INC	AC = AC + 1



Clear, Complement : E register

Hex	Mnemonic	RTL
7400	CLE	E = 0
7100	CME	E = ~ E

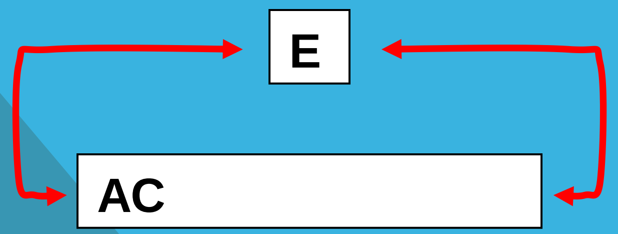


REGISTER REFERENCE INSTRUCTIONS

Shift AC register.

- This is a **circular** shift that is performed using the E register
- Control over timing ensures all operations operate in parallel
  - Eg. Use master-slave flip-flops in registers

Hex	Mnemonic	RTL
7080	CIR	AC(0-14) = AC(1-15), AC(15) = E, E = AC(0)
7040	CIL	AC(1-15) = AC(0-14), AC(0) = E, E = AC(15)



REGISTER REFERENCE INSTRUCTIONS

Skip on <condition> : AC register.

- Tests sign/value status of 2's complement integer in AC
- If status matches query, advance PC by one instruction word

Hex	Mnemonic	RTL
7010	SPA	(AC > 0) : PC = PC + 1
7008	SNA	(AC < 0) : PC = PC + 1
7004	SZA	(AC = 0) : PC = PC + 1

Skip on Zero condition : E register

- Test status of E bit. If not zero, proceed to the next instruction.
- If zero, advance PC by one instruction word (recall that it has already been incremented by one, so this causes skipping the next instruction in contiguous sequence.

Hex	Mnemonic	RTL
7002	SZE	~ E : PC = PC + 1

REGISTER REFERENCE INSTRUCTIONS

Halting the computer

- Disable all circuits (over-ride all specific Enable controls with a general Disable).

Hex	Mnemonic	RTL
7001	HLT	Disable all circuits

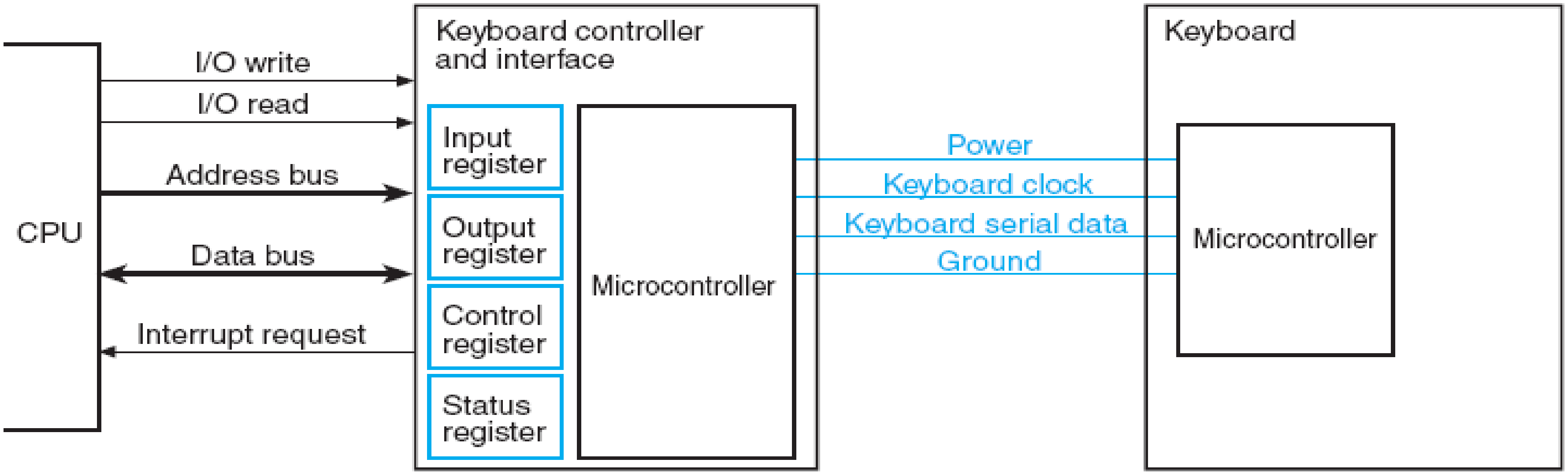
We have $D_7 I' T_3 = r$ (common to all register-reference instructions)

We have only one bit $IR(i) = B_i$ [bit in $IR(0—11)$ that specifies operation]

	r: $SC \leftarrow 0$	Clear *SC*
CLA	rB_{11}: $AC \leftarrow 0$	Clear *AC*
CLE	rB_{10}: $E \leftarrow 0$	Clear E
CMA	rB_9: $AC \leftarrow AC$	Complement AC
CME	rB_8: $E \leftarrow E$	Complement *E*
CIR	rB_7: $AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	rB_6: $AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$	Circulate left
INC	rB_5: $AC \leftarrow AC + 1$	Increment *AC*
SPA	rB_4: If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3: If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2: If $(AC = 0)$ then $(PC \leftarrow PC + 1)$	Skip if *AC* zero
SZE	rB_1: If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0: $S \leftarrow 0$ (*S* is a start—stop flip-flop)	Halt computer

INPUT, OUTPUT AND INTERRUPT INSTRUCTIONS

	15	14 12	11 0
IR	1	1 1 1	I/O operation control bits

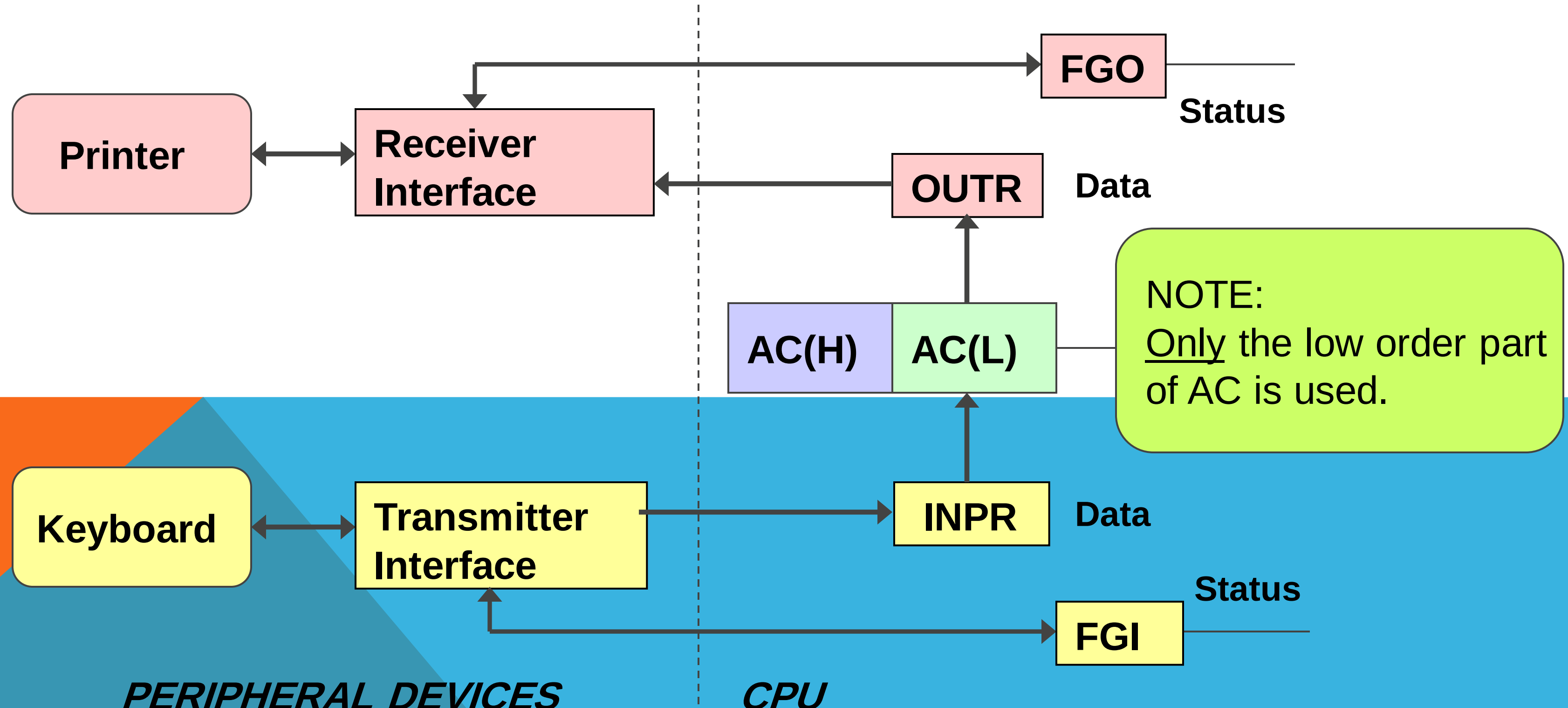


- The input register *INPR* consists of eight bits and holds alphanumeric input information.
- The 1-bit input flag *FGI* is a control flip-flop. The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer. The *flag* is needed to synchronize the timing rate difference between the input device and the computer.

INPUT/OUTPUT INSTRUCTIONS

Each peripheral device has a communications and control interface that interacts with the computer's interface logic circuits

- Input
  - Need a data buffer (INPR) and a flag (FGI) indicating buffer empty/full
- Output
  - Need a data buffer (OUTR) and a flag (FGO) indicating buffer empty/full



THE PROCESS OF INPUT INFORMATION TRANSFER IS AS FOLLOWS:

1. Initially, the input flag *FGI* is cleared to 0.
2. When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into *INPR* and the input flag *FGI* is set to 1. As long as the flag is set, the information in *INPR* cannot be changed by striking another key.
3. The computer checks the flag bit; if it is 1, the information from *INPR* is transferred in parallel into the eight low-order bits of *AC* and *FGI* is cleared to 0.
4. Once the flag is cleared, new information can be shifted into *INPR* by striking another key.

The output register *OUTR* works similarly but the direction of information flow is reversed.

THE PROCESS OF OUTPUT INFORMATION TRANSFER IS AS FOLLOWS:

1. **Initially, the output flag *FGO* is set to 1.**
2. **The computer checks the flag bit; if it is 1, the information in the eight least significant bits from *AC* is transferred in parallel to *OUTR* and *FGO* is cleared to 0.**
3. **The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets *FGO* to 1.**

The computer does not load a new character into *QUTR* when *FGQ* is 0 because this condition indicates that the output device is in the process of printing the character.

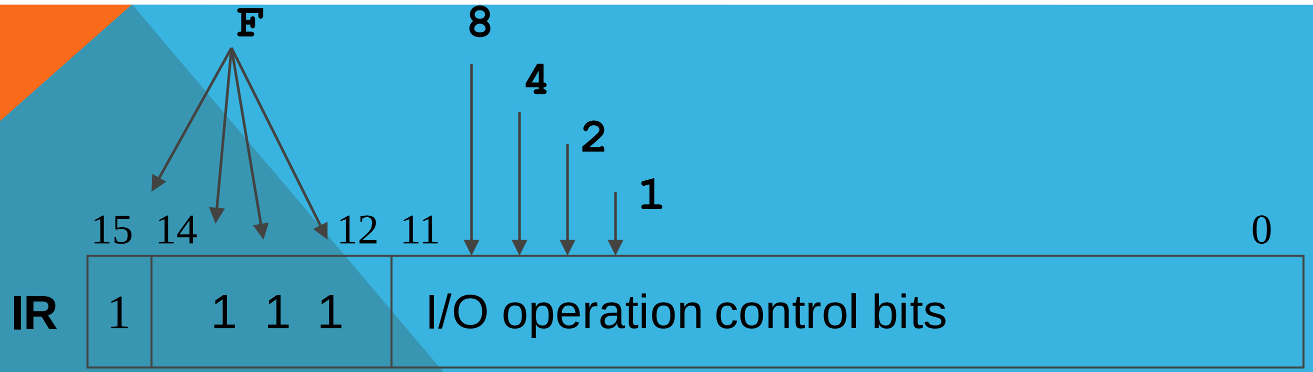
INPUT/OUTPUT INSTRUCTIONS

Used for communicating data between CPU and I/O peripheral devices

Also, need instructions to support programmed *polling*.

- Polling refers to waiting for a condition to be true before proceeding

16 bit OpCode	Mnemonic	Meaning
F800	INP	Input ASCII char
F400	OUT	Output ASCII char
F200	SKI	Skip if input flag (FGI=1)
F100	SKO	Skip if output flag (FGO=1)



INPUT-OUTPUT INSTRUCTIONS
INSTRUCTIONS STARTS WITH 0111

$D_7 I T_3$ = p (common to all input—output instructions)

$IR(i) = B_i$ [bit in $IR(6—11)$ that specifies the instruction]

	p: $SC \leftarrow 0$	Clear *SC*
INP	pB_{11}: $AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	*Input character*
OUT	PB_{10}: $OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	*Output character*
SKI	PB_9: $If(FGI=1)then(PC \leftarrow PC+1)$	Skip on input flag
SKO	PB_8: $If(FGO=1)then(PC \leftarrow PC+1)$	Skip on Output flag
ION	PB_7: IEN $\leftarrow$1	Interrupt enable on
IOF	PB_7 : IEN $\leftarrow$ 0	Interrupt enable off

INTERRUPTS

Input and Output interactions with electromechanical peripheral devices require huge processing times compared with CPU processing times

- I/O (milliseconds) versus CPU (nano/micro-seconds)

Interrupts permit other CPU instructions to execute while waiting for I/O to complete

- Need an additional 1-bit **IEN** flip-flop to store the interrupt status (0/1)

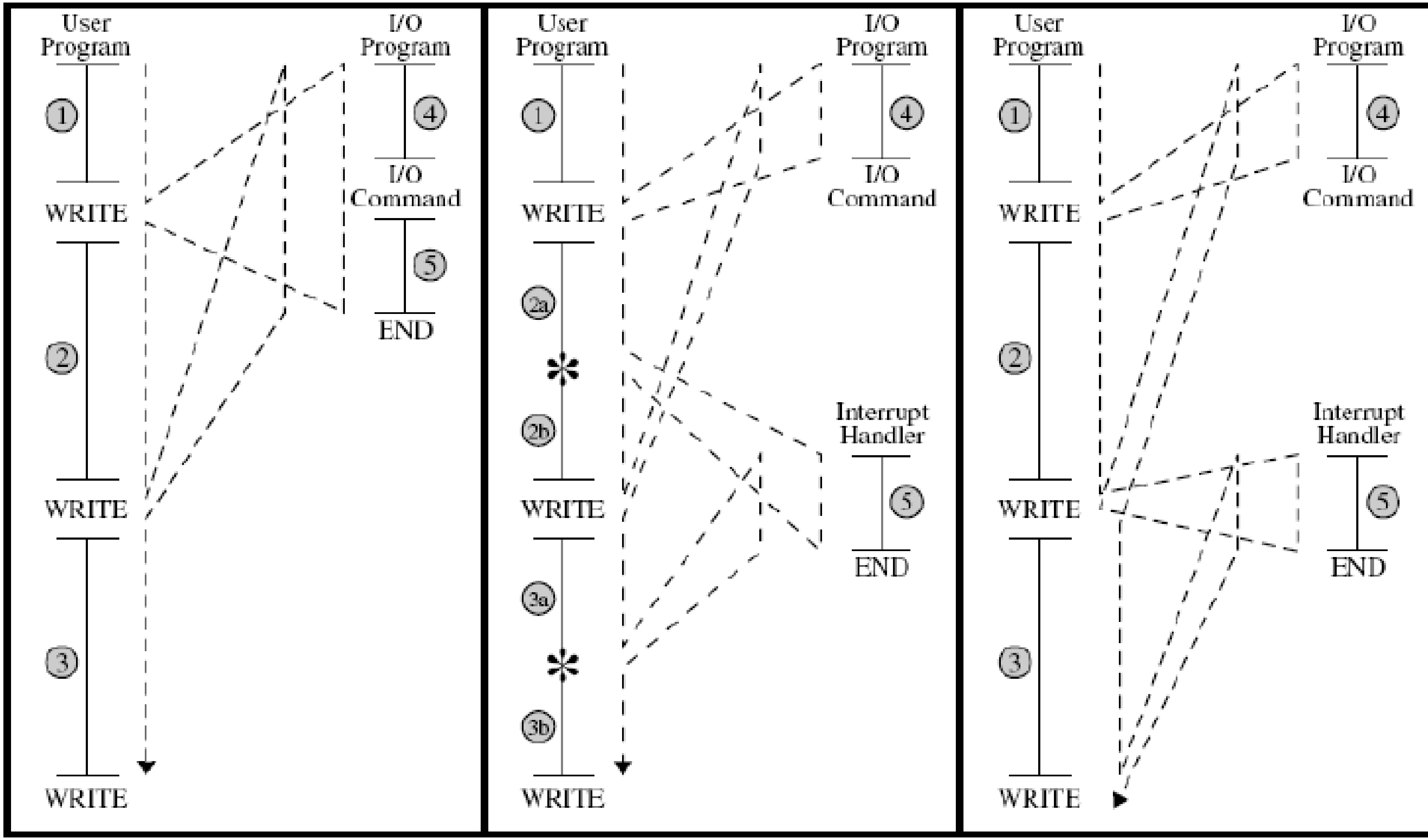
16 bit

OpCode	Mnemonic	Meaning
F080	ION	Interrupt Enabled (IEN ← 1)
F040	IOF	Interrupt Disabled (IEN ← 0)

IEN

	15	14–12	11–0
IR	1	1 1 1	I/O operation control bits

INPUT- OUTPUT AND PROGRAM INTERRUPT



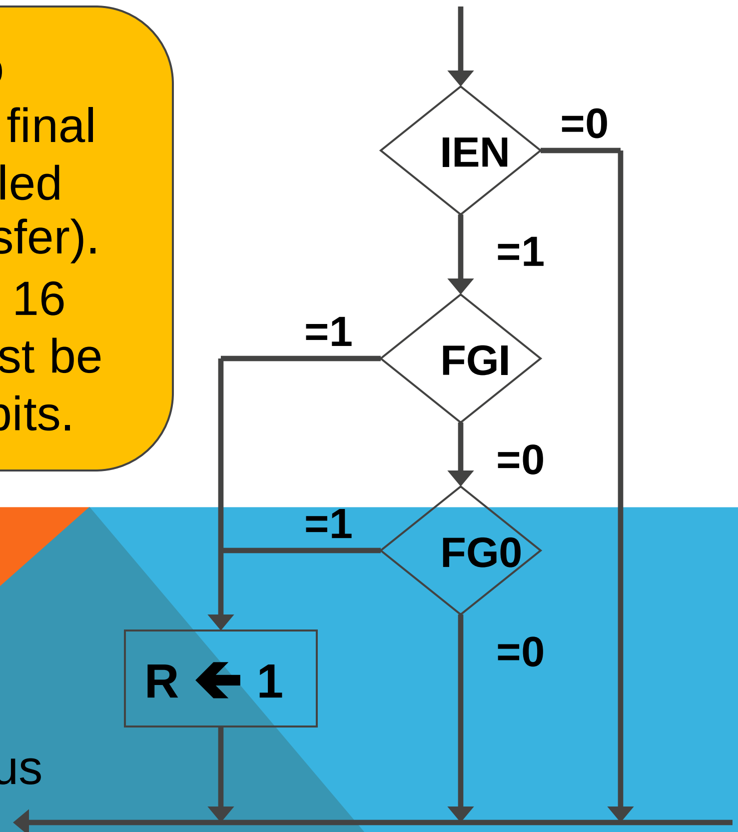
(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

INTERRUPTS

In this approach, interrupts are used only with I/O handling

- In addition to a flip-flop to store the Interrupt Enable state, one more flip-flop (R) is needed to store the I/O Status (Ready/Not_ready).
- In general, interrupts may be used with arbitrary instructions for exception trapping and handling

We will also require one final register, called **`TR`** (for transfer). This can be 16 bits, but must be at least 12 bits.



1-bit registers

IEN

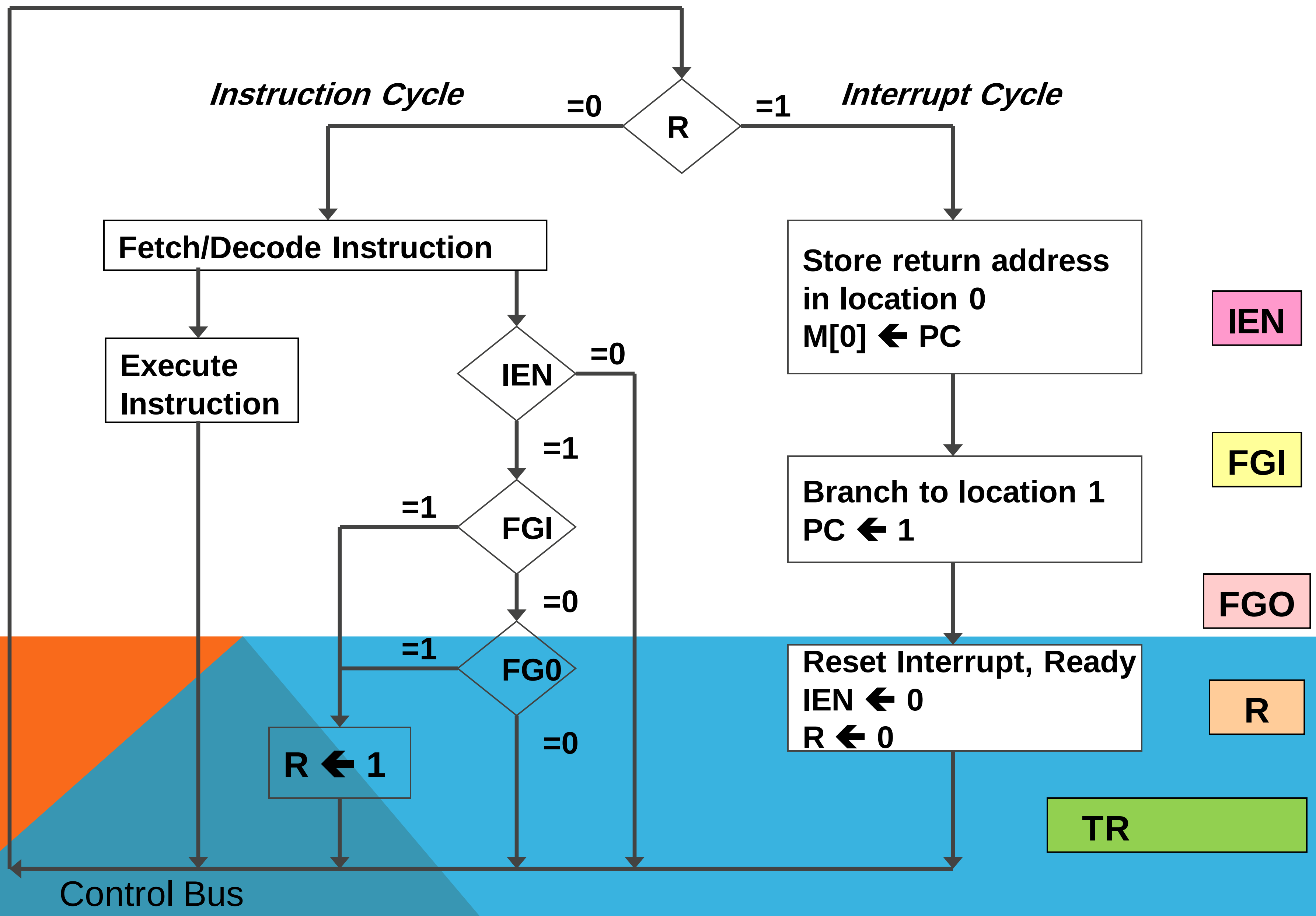
FGI

FGO

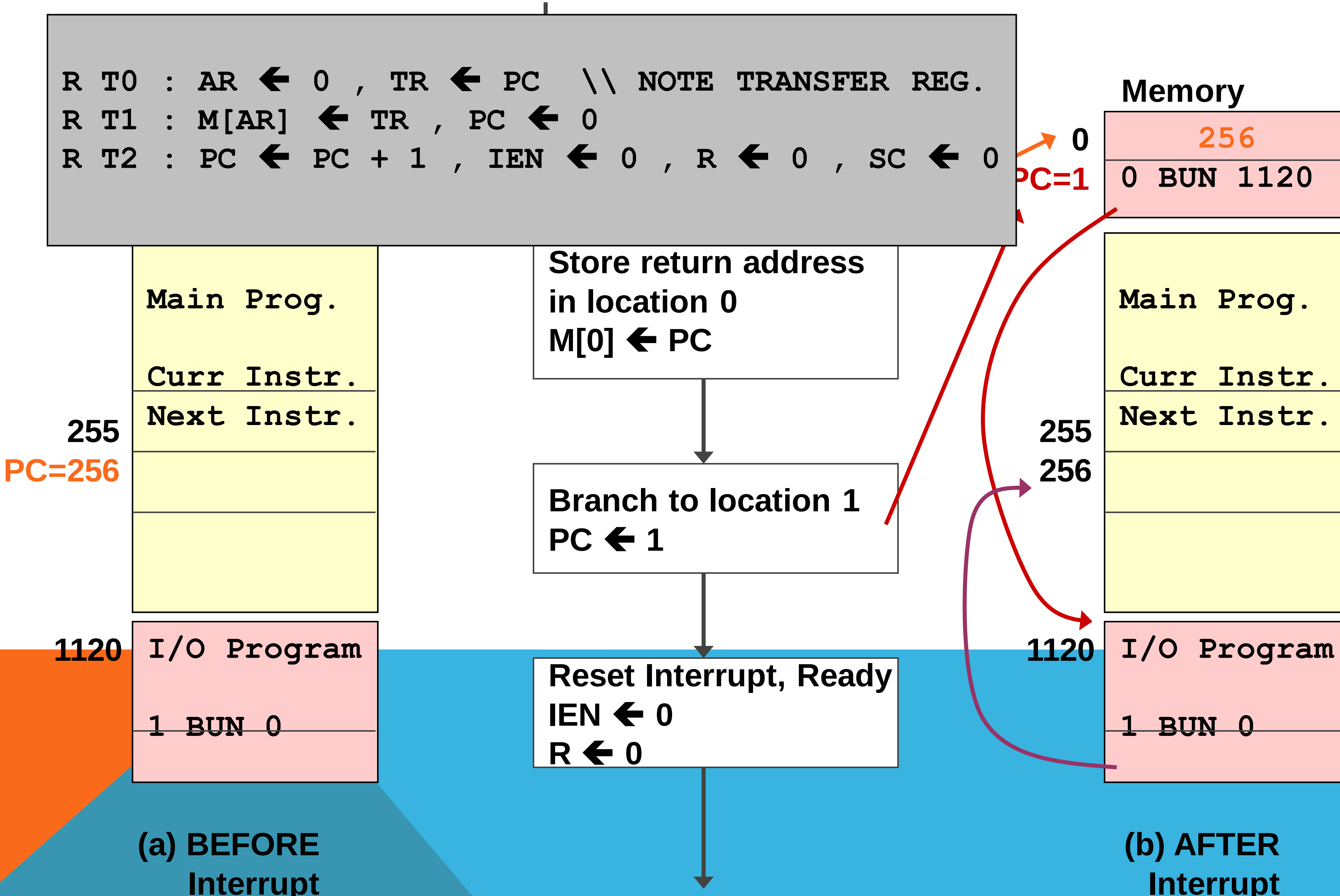
R

All of these flipflops are assumed to be reset to 0 when bootstrapping the computer.

INTERRUPT HANDLING FLOWCHART

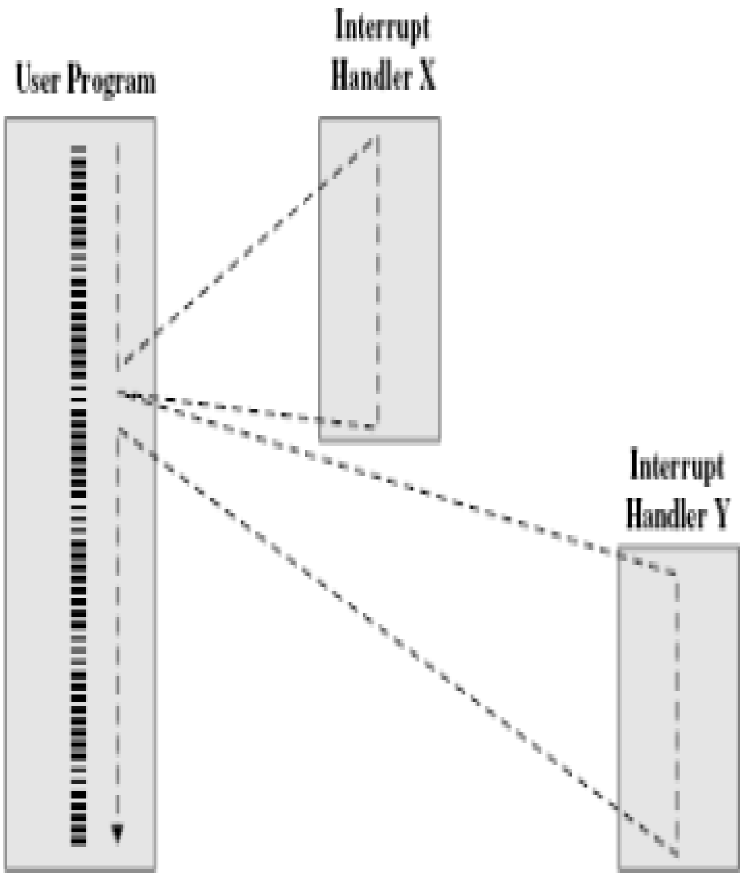


INTERRUPT HANDLING FLOWCHART

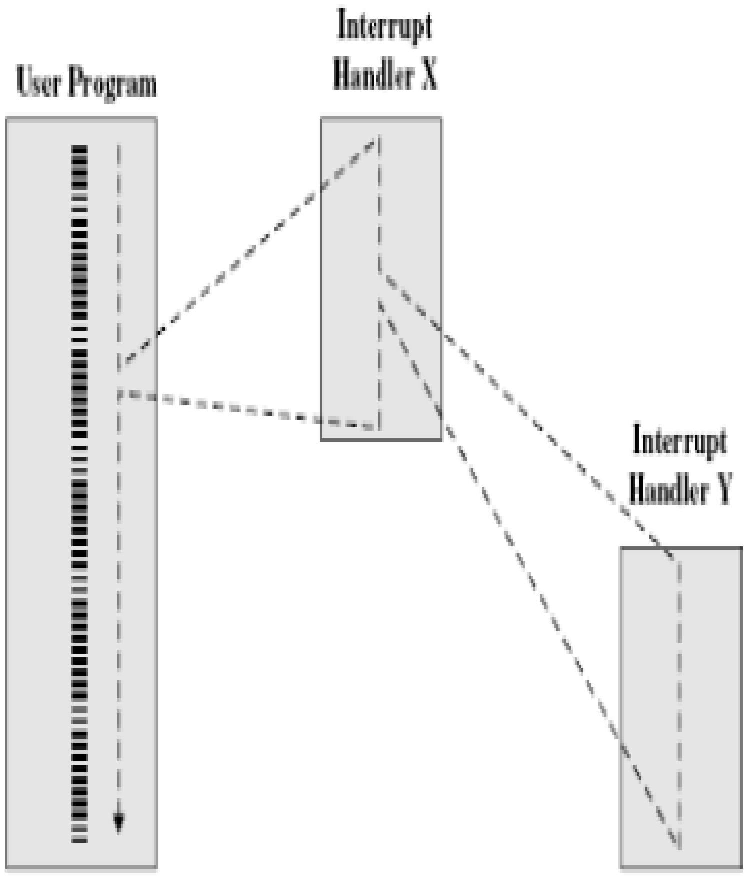


MULTIPLE INTERRUPTS

Multiple Interrupts - Sequential

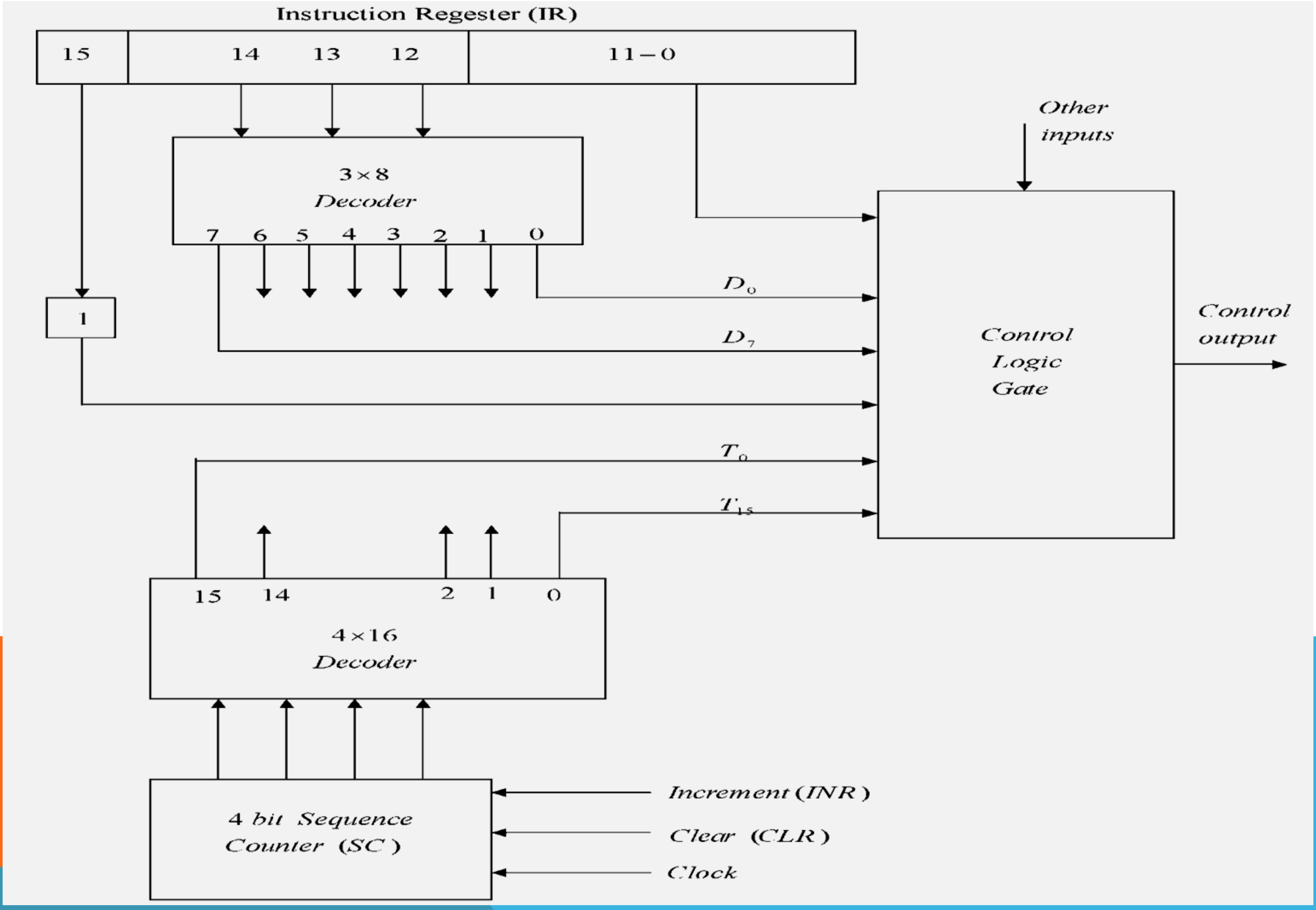


Multiple Interrupts – Nested



Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR],\quad PC \leftarrow PC+1$
Decode	$R'T_2$:	$D_0,\ldots,D_7 \leftarrow Decode\ IR(12-14),$
		$AR \leftarrow IR(0-11),\quad I \leftarrow IR(15)$
Indirect	$D_7'IT_3$:	$AR \leftarrow M[AR]$
Interrupt		
$T_0'T_1'T_2'(IEN)(FGI+FGO)$:		$R \leftarrow 1$
	RT_0 :	$AR \leftarrow 0,\quad TR \leftarrow PC$
	RT_1 :	$M[AR] \leftarrow TR,\quad PC \leftarrow 0$
	RT_2 :	$PC \leftarrow PC+1,\quad IEN \leftarrow 0,\quad R \leftarrow 0,\quad SC \leftarrow 0$
Memory reference		
AND	D_0T_4 :	$DR \leftarrow M[AR]$
	D_0T_5 :	$AC \leftarrow AC \wedge AR,\quad SC \leftarrow 0$
ADD	D_1T_4 :	$DR \leftarrow M[AR]$
	D_1T_5 :	$AC \leftarrow AC + AR,\quad E \leftarrow C_{OUT},\quad SC \leftarrow 0$
LDA	D_2T_4 :	$DR \leftarrow M[AR]$
	D_2T_5 :	$AC \leftarrow DR,\quad SC \leftarrow 0$
STA	D_3T_4 :	$M[AR] \leftarrow AC,\quad SC \leftarrow 0$
BUN	D_4T_4 :	$PC \leftarrow AR,\quad SC \leftarrow 0$
BSA	D_5T_4 :	$M[AR] \leftarrow PC,\quad AR \leftarrow AR+1$
	D_5T_5 :	$PC \leftarrow AR,\quad SC \leftarrow 0$
ISZ	D_6T_4 :	$DR \leftarrow M[AR]$
	D_6T_5 :	$DR \leftarrow DR+1$
	D_6T_6 :	$M[AR] \leftarrow DR,\quad if\ (DR=0) then (PC \leftarrow PC+1),\quad SC \leftarrow 0$
Register-reference		$D_7I'T_3 = r$ (*common to all register reference instruction*)
		$IR(i) = B_i\ (i = 0,1,2,\ldots,11)$
	r :	$SC \leftarrow 0$
CLA	rB_{11} :	$AC \leftarrow 0$
CLE	rB_{10} :	$E \leftarrow 0$
CMA	rB_9 :	$AC \leftarrow \overline{AC}$
CME	rB_8 :	$E \leftarrow \overline{E}$
CIR	rB_7 :	$AC \leftarrow shr AC,\quad AC(15) \leftarrow E,\quad E \leftarrow AC(0)$
CIL	rB_6 :	$AC \leftarrow shl AC,\quad AC(0) \leftarrow E,\quad E \leftarrow AC(15)$
INC	rB_5 :	$AC \leftarrow AC+1$
SPA	rB_4 :	$If\ (AC(15)=0) then (PC \leftarrow PC+1)$
SNA	rB_3 :	$If\ (AC(15)=1) then (PC \leftarrow PC+1)$
SZA	rB_2 :	$If\ (AC=0) then (PC \leftarrow PC+1)$
SZE	rB_1 :	$If\ (E=0) then (PC \leftarrow PC+1)$
HLT	rB_0 :	$S \leftarrow 0$
Input − output		$D_7IT_3 = p$ (*common to all register reference instruction*)
		$IR(i) = B_i\ (i = 6,7,8,9,10,11)$
	p :	$SC \leftarrow 0$
INP	pB_{11} :	$AC(0-7) \leftarrow INPR,\quad FGI \leftarrow 0$
OUT	pB_{10} :	$OUTR \leftarrow AC(0-7),\quad FGO \leftarrow 0$
SKI	pB_9 :	$If\ (FGI=1) then (PC \leftarrow PC+1)$
SKO	pB_8 :	$If\ (FGO=1) then (PC \leftarrow PC+1)$
ION	pB_7 :	$IEN \leftarrow 1$
IOF	pB_6 :	$IEN \leftarrow 0$

CONTROL LOGIC GATES



CONTROL OF REGISTERS: (AC)

- **Suppose that *we* want to derive the gate structure associated with the control inputs of *AR.* We scan Table 5-6 to find all *the* statements that change the content of *AR:***

  $R'T_0$: AR ← PC
  ***$R'T_2$:* .AR ← IR(0—11)**

  D'_7IT_3: AR←M[AR]
  ***RT_0:* AR ← 0**
  D_5T_4: AR←AR+1

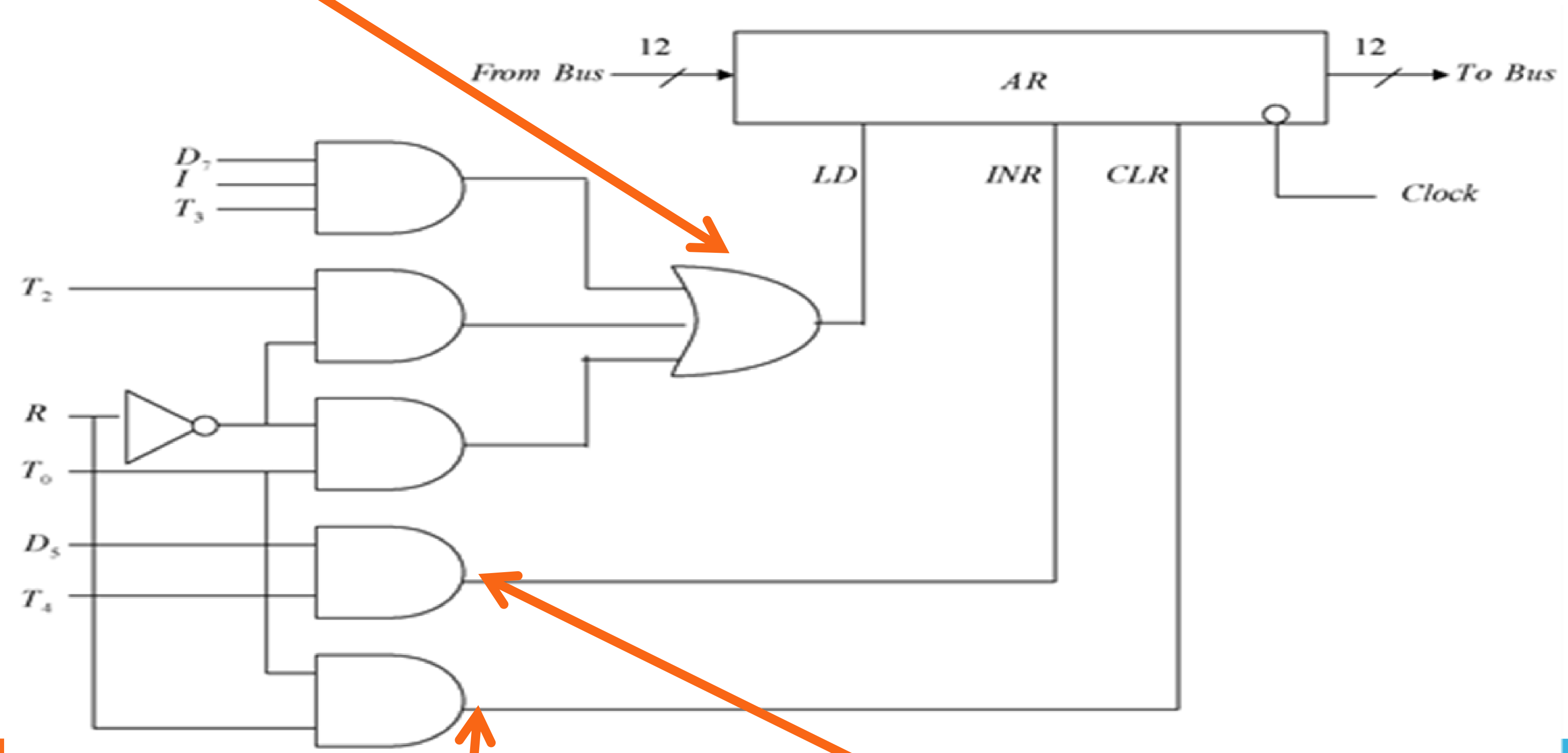
- **The control functions can be combined into three Boolean expressions as follows:**

LD (AR) = R'T0 + R'T2 + D'7IT3 / the load input of AR

CLR (AR) = RT0 / the clear input of AR

INR(AR) = D5T4 / the increment input of AR

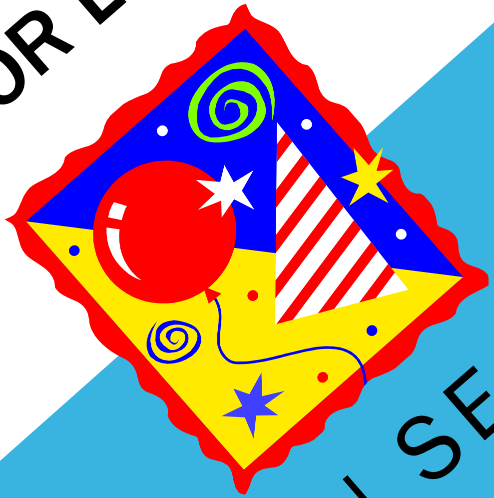
LD (AR) = R'T0 + R'T2 + D'7IT3



INR(AR) = D5T4

CLR (AR) = RT0

THANK YOU FOR LISTENING



SEE YOU IN SECOND GRADE – COMPUTER DEPARTMENT